Radio Transmission

Spoiler

First, notice that there is no way to distinguish the actual message from any of its cyclic shifts based on what is received from the station. For instance, in the example given in the task text, the 3-character message could be 'abc' or 'bca' or 'cab'.

This means that for a message of length L', the first L' characters received could be the message just as well as any other L' consecutive characters in the sequence. Since the message is repeated, this means each following character after that has to be the same as the one L' positions to the left. Such a correspondence can be found by comparing $S[1 \ldots L-i]$ to $S[i+1 \ldots L]$ for $i = 1, 2, \ldots, L$ and returning the smallest i for which they match. The running time of such a solution is $O(L^2)$.

The above algorithm is suspiciously similar to the naive method for finding a given pattern as a substring in a given text. This should serve as a good hint to look for similar optimizations. And indeed, the repeated comparisons look for the longest prefix of the received sequence that is also a suffix of the same sequence. Keeping track of such prefix/suffix pairs is exactly what the Knuth-Morris-Pratt algorithm ([**?**], [**?**]) is about. The solution based on the π-function from the Knuth-Morris-Pratt algorithm runs in $O(L)$ time using $O(L)$ memory in addition to the sequence itself.

One could also be tempted to take advantage of the search functions built into the standard library (`strstr` for C, either `strstr` or `string::find` for C++, `pos` for Pascal). As it turns out, they are not efficient enough; there are test cases where just one call to one of these functions would exceed the time limit.

References

[1] D. E. Knuth, J. H. Morris, V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Computing*, vol. 6, no. 2, pp. 323–350, 1977.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, Section 32.4: The Knuth-Morris-Pratt algorithm, pp. 923–931, MIT Press and McGraw-Hill, 2001.

Test data overview

1. $L = 1$, $S = \mathrm{a}$. $L' = 1$. Minimal test case. 4 points.
2. $L = 5$, $S = \mathrm{a}^5$. $L' = 1$. Minimal answer. 3 points.
3. $L = 5$, $S = \mathrm{abcde}$. $L' = 5$. "Maximal" answer. 3 points.
4. $L = 20$, $S = (\mathrm{abcdeabcdf})^2$. $L' = 10$. The message divides the sequence evenly. 10 points.
5. $L = 24$, $S = \mathrm{df}(\mathrm{abcdeabcdf})^2\mathrm{ab}$. $L' = 10$. The message does not divide the sequence evenly. 10 points.

6. $L = 100$, $S = \{\mathrm{xyz}\}^{100}$. $L' = 99$. A random sequence of 'x', 'y', 'z'. 10 points.
7. $L = 100000$, $S = x^5 a^{99990} x^5$. $L' = 99995$. A big test case engineered against the naive algorithm. 20 points.
8. $L = 200001$, $S = a^{100000} x a^{100000}$. $L' = 100001$. A big test case engineered against the naive algorithm. 20 points.
9. $L = 1000000$, $S = (\mathrm{a}^{16384}\mathrm{xa}^{32768}\mathrm{xa}^{1692}\mathrm{xa}^{32768}\mathrm{xa}^{16384})^{10}$, $L' = 100000$. Maximal test case engineered against the naive algorithm. 20 points.

In the above,

x^n	means 'x' repeated n times;
$(\mathrm{xyz})^n$	means 'xyz' repeated n times;
$\{\mathrm{xyz}\}^n$	means a sequence of n characters, each randomly taken from 'x', 'y', 'z'.